Contents

Collecting Metrics for TCP Services

4 minute read page test

This task shows how to configure Istio to automatically gather telemetry for TCP services in a mesh. At the end of this task, you can query default TCP metrics for your mesh.

The Bookinfo sample application is used as the example throughout this task.

Before you begin

- Install Istio in your cluster and deploy an application. You must also install Prometheus.
- This task assumes that the Bookinfo sample will be deployed in the `default` namespace. If you use a different namespace, update the example configuration and commands.

Collecting new telemetry data

1. Setup Bookinfo to use MongoDB.
    1. Install `v2` of the `ratings` service.

If you are using a cluster with automatic sidecar injection enabled, deploy the services using `kubectl`:

```
$ kubectl apply -f @samples/bookinfo/platf
orm/kube/bookinfo-ratings-v2.yaml@
serviceaccount/bookinfo-ratings-v2 created
deployment.apps/ratings-v2 created
```

If you are using manual sidecar injection, run the following command instead:

```
$ kubectl apply -f <(istioctl kube-inject
-f @samples/bookinfo/platform/kube/bookinf
o-ratings-v2.yaml@)
deployment "ratings-v2" configured
```

2. Install the `mongodb` service:

   If you are using a cluster with automatic sidecar injection enabled, deploy the services using `kubectl`:

```
$ kubectl apply -f @samples/bookinfo/platf
orm/kube/bookinfo-db.yaml@
service/mongodb created
deployment.apps/mongodb-v1 created
```

If you are using manual sidecar injection, run the following command instead:

```
$ kubectl apply -f <(istioctl kube-inject
-f @samples/bookinfo/platform/kube/bookinf
o-db.yaml@)
service "mongodb" configured
deployment "mongodb-v1" configured
```

3. The Bookinfo sample deploys multiple versions of each microservice, so begin by creating destination rules that define the service subsets corresponding to each version, and the load balancing policy for each subset.

```
$ kubectl apply -f @samples/bookinfo/netwo
rking/destination-rule-all.yaml@
```

If you enabled mutual TLS, run the following command instead:

```
$ kubectl apply -f @samples/bookinfo/networking/destination-rule-all-mtls.yaml@
```

To display the destination rules, run the following command:

```
$ kubectl get destinationrules -o yaml
```

Wait a few seconds for destination rules to propagate before adding virtual services that refer to these subsets, because the subset references in virtual services rely on the destination rules.

4. Create `ratings` and `reviews` virtual services:

```
$ kubectl apply -f @samples/bookinfo/networking/virtual-service-ratings-db.yaml@
virtualservice.networking.istio.io/reviews created
virtualservice.networking.istio.io/ratings created
```

2. Send traffic to the sample application.

   For the Bookinfo sample, visit `http://$GATEWAY_URL/productpage` in your web browser or use the following command:

   ```
   $ curl http://"$GATEWAY_URL/productpage"
   ```

   `$GATEWAY_URL` is the value set in the Bookinfo example.

3. Verify that the TCP metric values are being generated and collected.

   In a Kubernetes environment, setup

port-forwarding for Prometheus by using the following command:

```
$ istioctl dashboard prometheus
```

View the values for the TCP metrics in the Prometheus browser window. Select **Graph**. Enter the `istio_tcp_connections_opened_total` metric or `istio_tcp_connections_closed_total` and select **Execute**. The table displayed in the **Console** tab includes entries similar to:

```
istio_tcp_connections_opened_total{
destination_version="v1",
instance="172.17.0.18:42422",
job="istio-mesh",
canonical_service_name="ratings-v2",
canonical_service_revision="v2"}
```

```
istio_tcp_connections_closed_total{
destination_version="v1",
instance="172.17.0.18:42422",
job="istio-mesh",
canonical_service_name="ratings-v2",
canonical_service_revision="v2"}
```

Understanding TCP telemetry collection

In this task, you used Istio configuration to automatically generate and report metrics for all traffic to a TCP service within the mesh. TCP Metrics for all active connections are recorded every `15s` by default and this timer is configurable via `tcpReportingDuration`. Metrics for a connection are also recorded at the end of the connection.

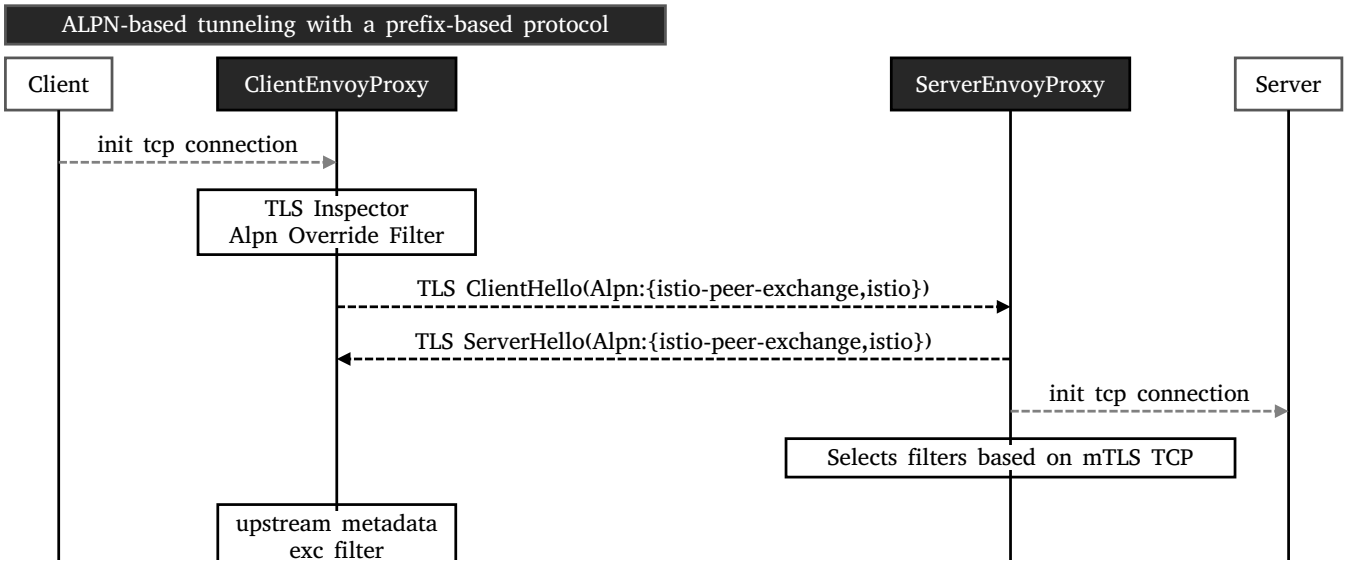
TCP attributes

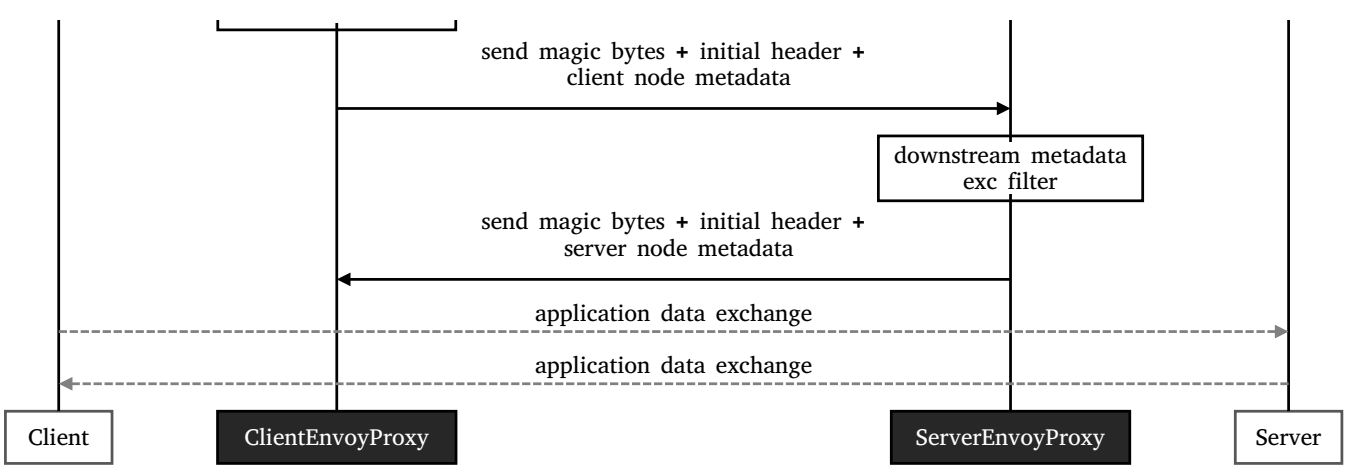
Several TCP-specific attributes enable TCP policy and control within Istio. These attributes are generated by Envoy Proxies and obtained from Istio using Envoy's Node Metadata. Envoy forwards Node Metadata to Peer Envoys using ALPN based tunneling and a prefix based protocol. We define a new protocol `istio-peer-exchange`, that is advertised and prioritized by the client and the server sidecars in the mesh. ALPN negotiation resolves the protocol to `istio-peer-exchange` for connections between Istio enabled proxies, but not between an Istio enabled proxy and any other proxy. This protocol extends TCP as follows:

1. TCP client, as a first sequence of bytes, sends a magic byte string and a length

prefixed payload.

2. TCP server, as a first sequence of bytes, sends a magic byte sequence and a length prefixed payload. These payloads are protobuf encoded serialized metadata.
3. Client and server can write simultaneously and out of order. The extension filter in Envoy then does the further processing in downstream and upstream until either the magic byte sequence is not matched or the entire payload is read.





TCP Attribute Flow

Cleanup

- Remove the `port-forward` process:

```
$ killall istioctl
```

- If you are not planning to explore any follow-on tasks, refer to the Bookinfo cleanup instructions to shutdown the application.